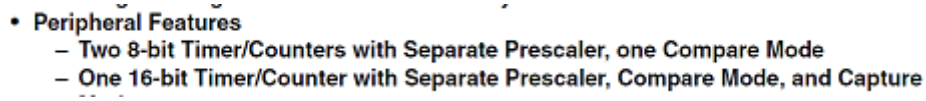
Таймеры-счетчики. Прерывания

Таймеры-счётчики - это такие устройства или модули в микроконтроллере, которые, как видно из названия, постоянно что-то считают. Считают они либо до определённой величины, либо до такой величины, сколько они битности. Считают они постоянно с одной скоростью, со скоростью тактовой частоты микроконтроллера, поправленной на делители частоты, которые мы будем конфигурировать в определённых регистрах.

И вот эти таймеры-счётчики постоянно считают, если мы их инициализируем.

Таймеров в МК **Atmega8** три.


- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler, one Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture

Два из них — это **8-битные** таймеры, то есть такие, которые могут максимально досчитать только до 255. Данной величины нам будет маловато. Даже если мы применим максимальный делитель частоты, то мы не то что секунду не отсчитаем, мы даже полсекунды не сможем посчитать. А у нас задача именно такая, чтобы досчитывать до 1 секунды, чтобы управлять наращиванием счёта светодиодного индикатора. Можно конечно применить ещё наращивание переменной до определенной величины, но хотелось бы полностью аппаратного счёта.

Но есть ещё один таймер — это полноправный **16-битный** таймер. Он не только **16-битный**, но есть в нём ещё определённые прелести, которых нет у других таймеров. С данными опциями мы познакомимся позже.

Вот этот 16-битный таймер мы и будем сегодня изучать и использовать. Также, познакомившись с данным таймером, вам ничего не будет стоить самостоятельно изучить работу двух других, так как они значительно проще. Но тем не менее 8-битные таймеры в дальнейшем мы также будем рассматривать, так как для достижения более сложных задач нам одного таймера будет недостаточно.

Теперь коротко о прерываниях.

Прерывания (**Interrupts**) — это такие механизмы, которые прерывают код в зависимости от определённых условий или определённой обстановки, которые будут диктовать некоторые устройства, модули и шины, находящиеся в микроконтроллере.

В нашем контроллере **Atmega8** существует **19 видов прерываний**. Вот они все находятся в таблице в технической документации на контроллер

Table 18. Reset and Interrupt Vectors

Vector No.	Program Address[2]	Source	Interrupt Definition
1	0x000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset, and Watchdog Reset
2	0x001	INT0	External Interrupt Request 0
3	0x002	INT1	External Interrupt Request 1
4	0x003	TIMER2 COMP	Timer/Counter2 Compare Match
5	0x004	TIMER2 OVF	Timer/Counter2 Overflow
6	0x005	TIMER1 CAPT	Timer/Counter1 Capture Event
7	0x006	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	0x007	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	0x008	TIMER1 OVF	Timer/Counter1 Overflow
10	0x009	TIMER0 OVF	Timer/Counter0 Overflow
11	0x00A	SPI, STC	Serial Transfer Complete
12	0x00B	USART, RXC	USART, Rx Complete
13	0x00C	USART, UDRE	USART Data Register Empty
14	0x00D	USART, TXC	USART, Tx Complete
15	0x00E	ADC	ADC Conversion Complete
16	0x00F	EE_RDY	EEPROM Ready
17	0x010	ANA_COMP	Analog Comparator
18	0x011	TWI	Two-wire Serial Interface
19	0x012	SPM_RDY	Store Program Memory Ready

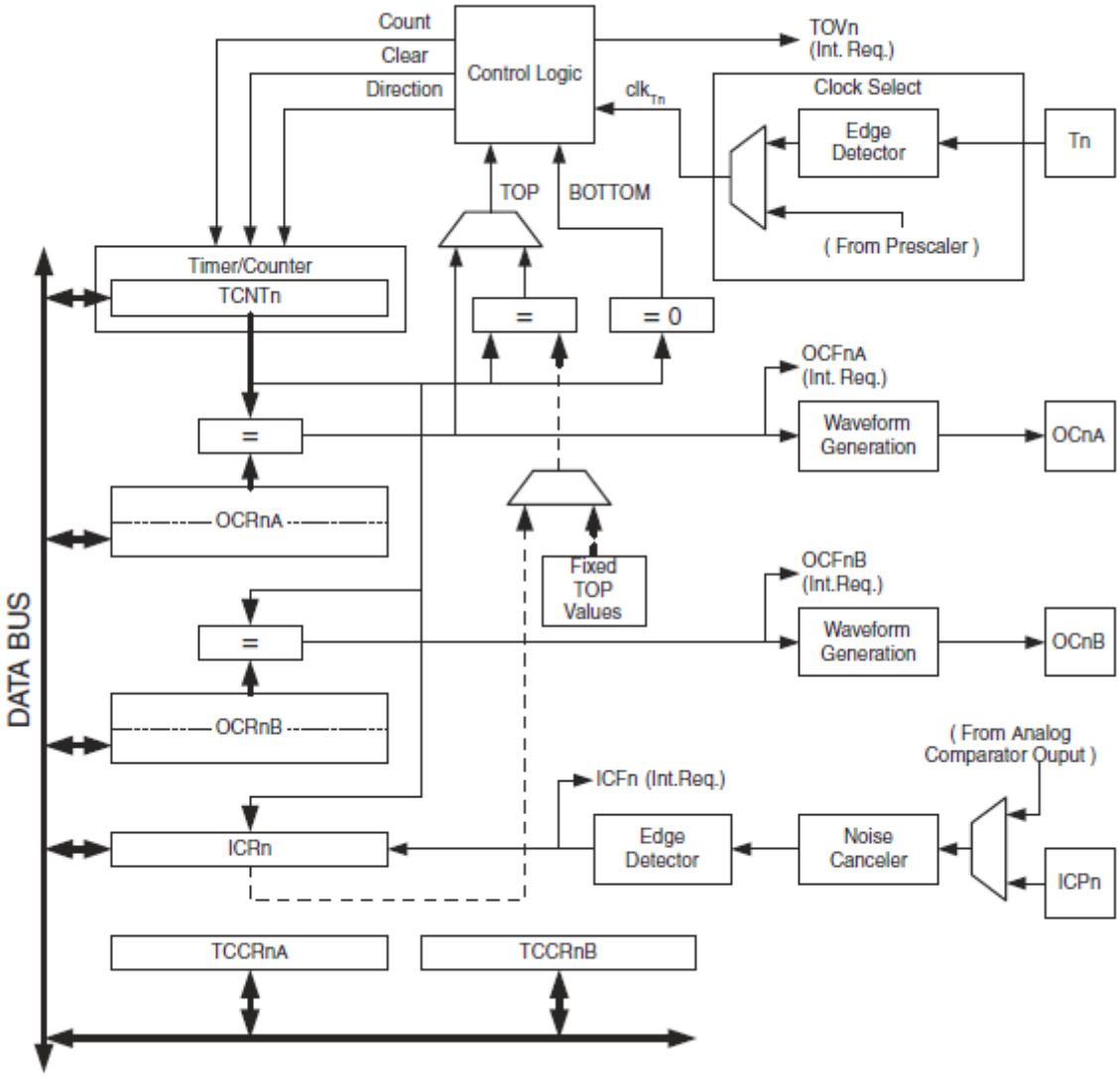
Какого типа могут быть условия? В нашем случае, например, досчитал таймер до определённой величины, либо, например, в какую-нибудь шину пришёл байт и другие условия.

На данный момент мы будем обрабатывать прерывание, которое находится в таблице, размещённой выше на 7 позиции — **TIMER1 COMPA**, вызываемое по адресу 0x006.

Теперь давайте рассмотрим наш 16-битный таймер или **TIMER1**.

Вот его структурная схема:

Figure 32. 16-bit Timer/Counter Block Diagram[1]



Мы видим там регистр **TCNTn**, в котором постоянно меняется число, т.е. оно постоянно наращивается. Практически это и есть счётчик или регистр, который хранит число, до которого и досчитал таймер.

А в регистры **OCRnA** и **OCRnB** (буквы n — это номер таймера, в нашем случае будет 1) — это регистры, в которые мы заносим число, с которым будет сравниваться число в регистре **TCNTn**.

Например, занесли мы какое-нибудь число в регистр **OCRnA** и как только данное число совпало со значением в регистре счёта, то возникнет прерывание, и мы его сможем обработать. Таймеры с прерываниями очень похожи на обычную задержку в коде, только когда мы находимся в задержке, то мы в это время не можем выполнять никакой код (ну опять же образно "мы", на самом деле АЛУ). А когда считает таймер, то весь код нашей программы в это время спокойно выполняется. Так что мы выигрываем колоссально, не

давая простаивать огромным ресурсам контроллера по секунде или даже по полсекунды. В это время мы можем обрабатывать нажатия кнопок, которые мы также можем обрабатывать в таймере и многое другое.

Есть также регистр **TCCR**. Данный регистр — это регистр управления. Там настраиваются определенные биты, отвечающие за конфигурацию таймера.

Также у таймера существует несколько режимов, с которыми мы также познакомимся немного позднее.

Он состоит из двух половинок, так как у нас контроллер 8-битный и в нем не может быть 16-битных регистров. Поэтому в одной половинке регистра (а физически в одном регистре) хранится старшая часть регистра, а в другом — младшая. Можно также назвать это регистровой парой, состоящей из двух отдельных регистров **TCCR1A** и **TCCR1B**. Цифра 1 означает то, что регистр принадлежит именно таймеру 1.

Данный регистр **TCCR** отвечает за установку делителя, чтобы таймер не так быстро считал, также он отвечает (вернее его определённые биты) за установку определённого режима.

За установку режима отвечают биты WGM:

Table 39. Waveform Generation Mode Bit Description

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation[1]	TOP	Update of OCR1x	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Мы видим здесь очень много разновидностей режимов:

Normal — это обычный режим, таймер считает до конца.

PWM — это **ШИМ** только разные разновидности, то есть таймер может играть роль *широтно-импульсного модулятора*. С данной технологией мы будем знакомиться в более поздних занятиях.

CTC — это сброс по совпадению, как раз то что нам будет нужно. Здесь то и сравниваются регистры TCNT и OCR. Таких режима два, нам нужен первый, второй работает с другим регистром.

Создадим первую функцию и назовём **timer_ini:**

```
//————————————————————
void timer_ini(void)
{


}
//————————————————————
```

Давайте наши функции и какие-то законченные блоки с объявлением глобальных переменных, с прототипами функций будем отделять друг от друга вот такими чёрточками, которые за счет наличия двух слешей впереди компилятор обрабатывать не будет и примет их за комментарии. За счёт этих отчерчиваний мы будем видеть, где заканчивается одна функция и начинается другая.

Данная функция, как мы видим не имеет ни каких аргументов — ни входных, не возвращаемых. Давайте сразу данную функцию вызовем в функции main():

```
unsigned char butcount=0, butstate=0;
timer_ini();
```

Теперь мы данную функцию начнём наполнять кодом.

Начнем с регистра управления таймером, например, с **TCCR1B**. Используя нашу любимую операцию "ИЛИ", мы в определённый бит регистра занесём единичку:

```
void timer_ini(void)
{
  TCCR1B |= (1<<WGM12); // устанавливаем режим CTC (сброс по совпадению)
```

Из комментария мы видим, что мы работает с битами режима, и установим мы из них только бит **WGM12**, остальные оставим нули. Исходя из этого мы сконфигурировали вот такой режим:

4	0	1	0	0	CTC	OCR1A	Immediate	MAX

Также у таймера существует ещё вот такой регистр — **TIMSK**. Данный регистр отвечает за маски прерываний — **Interrupt Mask**. Доступен данный регистр для всех таймеров, не только для первого, он общий. В данном регистре мы установим бит **OCIE1A**, который включит нужный нам тип прерывания **TIMER1 COMPA**

- **Bit 4 – OCIE1A: Timer/Counter1, Output Compare A Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Output Compare A match interrupt is enabled. The corresponding Interrupt Vector (see "Interrupts" on page 46) is executed when the OCF1A Flag, located in TIFR, is set.

```
  TCCR1B |= (1<<WGM12); // устанавливаем режим CTC (сброс по совпадению)
  TIMSK |= (1<<OCIE1A); //устанавливаем бит разрешения прерывания 1ого счетчика по совпадению с OCR1A(H и L)
```

Теперь давайте рассмотрим сами регистры сравнения **OCR1A(H и L)**. Для этого придётся немного посчитать. Регистр **OCR1AH** хранит старшую часть числа для сравнения, а регистр **OCR1AL** — младшую.

Но прежде чем посчитать, давайте пока напишем код с любыми значениями данного регистра и потом поправим, так как дальше мы будем инициализировать делитель, и он тоже будет участвовать в расчёте требуемого времени счёта. Без делителя таймер будет слишком быстро считать.

```
  TIMSK |= (1<<OCIE1A); //устанавливаем бит разрешения прерывания 1ого счетчика по совпадению с OCR1A(H и L)
```

OCR1AH = 0b10000000; //записываем в регистр число для сравнения

OCR1AL = 0b00000000;

TCCR1B |= (); //установим делитель.

Пока никакой делитель не устанавливаем, так как мы его ещё не посчитали. Давайте мы этим и займёмся.

Пока у нас в регистре **OCR1A** находится число **0b1000000000000000**, что соответствует десятичному числу **32768.**

Микроконтроллер у нас работает, как мы договорились, на частоте 8000000 Гц.

Разделим **8000000** на **32768**, получим приблизительно **244,14**. Вот с такой частотой в герцах и будет работать наш таймер, если мы не применим делитель. То есть цифры наши будут меняться **244 раза в секунду**, поэтому мы их даже не увидим. Поэтому нужно будет применить делитель частоты таймера. **Выберем делитель на 256**. Он нам как раз подойдёт, а ровно до 1 Гц мы скорректируем затем числом сравнения.

Вот какие существуют делители для 1 таймера:

Table 40. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge
1	1	1	External clock source on T1 pin. Clock on rising edge

Выделим в таблице требуемый нам делитель. Мы видим, что нам требуется установить только бит **CS12.**

Так как делитель частоты у нас **256**, то на этот делитель мы поделим **8000000**, получится **31250**, вот такое вот мы и должны занести число в **TCNT.** До такого числа и будет считать наш таймер, чтобы досчитать до 1 секунды. **Число 31250** — это в двоичном представлении **0b0111101000010010.**

Занесём данное число в регистровую пару, и также применим делитель:

OCR1AH = 0b**01111010**; //записываем в регистр число для сравнения

OCR1AL = 0b**00010010**;

TCCR1B |= (**1<<CS12**); //установим делитель.

С данной функцией всё.

Теперь следующая функция — обработчик прерывания от таймера по совпадению. Пишется она вот так

ISR (TIMER1_COMPA_vect)

{

}

И тело этой функции будет выполняться само по факту наступления совпадения чисел.

Нам нужна будет переменная. Объявим её глобально, в начале файла:

#include <util/delay.h>

//———————————————

unsigned char i;

//———————————————

Соответственно, из кода в функции main() мы такую же переменную уберём:

int main(void)

{

unsigned char i;

Также закомментируем весь код в бесконечном цикле. Его роль теперь у нас будет выполнять таймер, и, я думаю, он с этим справится не хуже, а даже лучше, "никому" при этом не мешая.

while(1)

```
{
// for(i=0;i<10;i++)
// {
//   while (butstate==0)
//   {
//     if (!(PINB&0b00000001))
//     {
//       if(butcount < 5)
//       {
//         butcount++;
//       }
//       else
//       {
//         i=0;
//         butstate=1;
//       }
//     }
//     else
//     {
//       if(butcount > 0)
//       {
//         butcount—;
//       }
//       else
//       {
//         butstate=1;
//       }
//     }
//   }
//   segchar(i);
//   _delay_ms(500);
//   butstate=0;
// }
```

```
}
```

Теперь, собственно, тело функции-обработчика. Здесь мы будем вызывать функцию segchar. Затем будем наращивать на 1 переменную **i**. И чтобы она не ушла за пределы однозначного числа, будем её обнулять при данном условии:

```
ISR (TIMER1_COMPA_vect)
{
 if(i>9) i=0;
 segchar(i);
 i++;
}
```

Теперь немного исправим код вначале функции main(). Порт **D**, отвечающий за состояние сегментов, забьём единичками, чтобы при включении у нас не светился индикатор, так как он с общим анодом. Затем мы здесь занесём число 0 в глобальную переменную i, просто для порядка. Вообще, как правило, при старте в неинициализированных переменных и так всегда нули. Но мы всё же проинициализируем её. И, самое главное, чтобы прерывание от таймера работало, её недостаточно включить в инициализации таймера. Также вообще для работы всех прерываний необходимо разрешить глобальные прерывания. Для этого существует специальная функция **sei() — Set Interrupt**.

Теперь код будет вот таким:

```
DDRB = 0x00;
PORTD = 0b11111111;
PORTB = 0b00000001;
i=0;
sei();
while(1)
```

Также ещё мы обязаны подключить файл библиотеки прерываний вначале файла:

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
```

Также переменные для кнопки нам пока не потребуются, так как с кнопкой мы сегодня работать не будем. Закомментируем их.

```
int main(void)
{
//unsigned char butcount=0, butstate=0;
timer_ini();
```

Соберём наш код и проверим его работоспособность в Proteus.

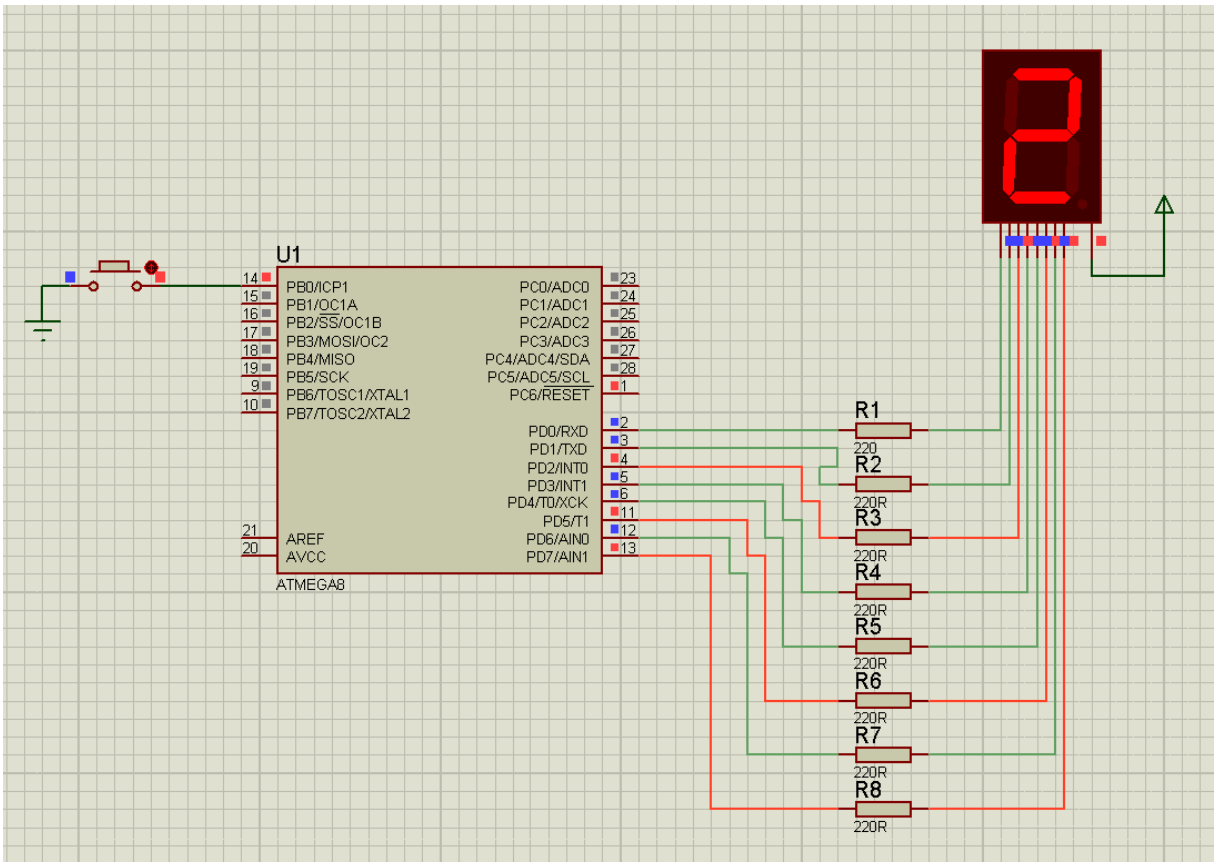
Полный код:

```
#define F_CPU 8000000L
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

//-----------------------------------------
unsigned char i;
//-----------------------------------------
void segchar (unsigned char seg)
{
	switch(seg)
	{
		case 1: PORTD = 0b11111001; break;
		case 2: PORTD = 0b10100100; break;
		case 3: PORTD = 0b10110000; break;
		case 4: PORTD = 0b10011001; break;
		case 5: PORTD = 0b10010010; break;
		case 6: PORTD = 0b10000010; break;
		case 7: PORTD = 0b11111000; break;
		case 8: PORTD = 0b10000000; break;
		case 9: PORTD = 0b10010000; break;
		case 0: PORTD = 0b11000000; break;
	}
}
//-----------------------------------------
void timer_ini(void)
{
	TCCR1B |= (1<<WGM12);	// устанавливаем режим CTC (сброс по совпадению)
```

```
	TIMSK |= (1<<OCIE1A);	//устанавливаем бит разрешения прерывания 1ого счетчика по совпадению с OCR1A(H и L)
	OCR1AH = 0b01111010;	//записываем в регистр число для сравнения
	OCR1AL = 0b00010010;
	TCCR1B |= (1<<CS12);	//установим делитель.
}
//-----------------------------------------
ISR (TIMER1_COMPA_vect)
{
	if(i>9) i=0;
	segchar(i);
	i++;
}
//-----------------------------------------
int main(void)
{
	//unsigned char butcount=0, butstate=0;
	timer_ini();
	DDRD = 0xFF;
	DDRB = 0x00;
	PORTD = 0b11111111;
	PORTB = 0b00000001;
	i=0;
	sei();

	while(1)
	{
		//		for(i=0;i<10;i++)
		//		{
		//			while (butstate==0)
		//			{
		//				if (!(PINB&0b00000001))
		//				{
		//					if(butcount < 5)
		//					{
		//						butcount++;
		//					}
		//					else
		//					{
		//						i=0;
		//						butstate=1;
		//					}
		//				}
		//				else
```

```
                //                          {
                //                                  if(butcount > 0)
                //                                  {
                //                                          butcount--;
                //                                  }
                //                                  else
                //                                  {
                //                                          butstate=1;
                //                                  }
                //                          }
                //                  }
                //                  segchar(i);
                //                  _delay_ms(500);
                //                  butstate=0;
                //          }
        }
}
```

Работа в среде Proteus:



Задание на выполнение:

1. Составить конспект по данному материалу о работе таймеров-счетчиков.
2. Изменить представленную программу для работы с портом B.
3. Какую функцию выполняет **segchar()**?

Результаты выполнения работы отправить на e-mail: rasov@rambler.ru с темой **Таймеры_ФИО**